WL-TR-94-5046

A KEY ELEMENT TOWARD CONCURRENT ENGINEERING OF HARDWARE AND SOFTWARE: BINDING VERY HIGH SPEED INTEGRATED CIRCUITS (VHSIC) HARDWARE DESCRIPTION LANGUAGE (VHDL) WITH ADA 95

MICHAEL T. MILLS, LT COL

OCTOBER 1994

FINAL REPORT FOR 04/12/94-10/24/94

SOLID STATE ELECTRONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7331

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that government may have formulated or in any way supplied the said drawings, specifications, or otherwise in any manner constructed, as in licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Michael T. Mills

MICHAEL T. MILLS, LtCol, USAFR
Design Branch
Microelectronics Division

John W Hines

JOHN W. HINES, Chief
Design Branch
Microelectronics Division

Stanley E Wagner

STANLEY E. WAGNER, Chief
Microelectronics Division
Solid State Electronics Directorate

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify WL/ELED, WPAFB, OH 45433-7331 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCT 1994	FINAL 04/12/94-10/24/94

4. TITLE AND SUBTITLE A KEY ELEMENT TOWARD CONCURRENT ENGINEERING OF HARDWARE AND SOFTWARE: BINDING VERY HIGH SPEED INTEGRATED CIRCUITS (VHSIC) HARDWARE DESCRIPTION LANGUAGE (VHDL) WITH ADA 95

5. FUNDING NUMBERS

6. AUTHOR(S)

MICHAEL T. MILLS, LT COL

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SOLID STATE ELECTRONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7331

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SOLID STATE ELECTRONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7331

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

WL-TR-94-5046

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes a software interface (or binding) for supporting concurrent development of electronic hardware designed in Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) with software programmed in Ada 95. The purpose of this binding is to provide an initial framework from which future computer aided engineering (CAE) implementors can develop concurrent engineering capability into their work stations. This software binding includes VHDL calls to Ada 95 subprograms and functions using the new VHDL-93 'FOREIGN attribute and providing optional Ada 95 functionality in place of concurrent VHDL process statements to meet Ada 95 language rules expected by future Ada 95 compilers. A binding is also provided for Ada 95 calls to VHDL representations using new Ada 95 pragmas: Import, Export, and Convention. BNF descriptions were expanded to analyze constraint differences for consistent functionality across languages. Examples show VHDL processes synchronizing with Ada 95 tasks and Ada 95 remote procedure calls across partitions for distributed processing applications.

14. SUBJECT TERMS

VHDL - ANSI/IEEE Std 1076-1993 - Design Language - Hardware Description Language, Ada 95 - ANSI/ISO revision of ANSI/MIL STD 1815A, VHDL to Ada 95 Binding

15. NUMBER OF PAGES 29

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. *Introduction*

This report describes software bindings for concurrent development of electronic hardware designed in Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) [1] with software programmed in Ada 95 [2]. The purpose of these bindings are to provide a framework from which future computer aided engineering (CAE) implementors can develop concurrent engineering capability into their work stations. These software bindings include VHDL calls to Ada 95 subprograms and functions using the new VHDL-93 'FOREIGN attribute. The called procedures written in Ada 95 can provide functionality similar to alternate VHDL processes. Bindings are also provided for Ada 95 calls to VHDL representations using new Ada 95 pragmas: Import, Export, and Convention. The latest revision of VHDL and draft Ada 95 were selected to make these bindings as current as possible.

These bindings are not only proposed to pass values between VHDL and Ada 95 designs but also can be used to pass control parameters to synchronize VHDL processes with Ada tasks and other Ada 95 concurrent mechanisms. Control parameters are also proposed to provide synchronous and asynchronous control between VHDL processes and Ada 95 remote procedure calls across partitions for distributed processing. The system designer can use this methodology to simulate prototypes of different allocations of hardware processes vs. concurrent software features.

In order to compare detailed VHDL and similar Ada 95 structures for language rule differences, BNF descriptions were combined into expanded descriptions. These descriptions plus semantic rules from the two standards were used to compare the behaviors of VHDL processes with Ada 95 tasks, subroutines, packages, basic declarations and types, statements, and VHDL entities with Ada 95 generic packages. The goal was to maintain consistent functionality across the VHDL and Ada 95 language boundaries.

2. *Rationale for Binding VHDL with Ada 95*

A binding of two programming languages provides the capability for a program written in one language to call subprograms existing in a program of another language. Bindings are normally required to handle non-compatible parameter passing conventions, syntax differences in the actual parameters of the subprogram call, syntax differences in the structure of the subprogram call, or semantic (language rule) differences which could restrict subprogram results from what is expected in the calling program. Bindings have become very important requirements for reusing large amounts of existing code in languages other than what is currently used.

The purpose of this report is to provide bindings to serve as gateways for VHDL implemented designs to call behaviors programmed in Ada 95 being developed concurrently. These bindings, if implemented by computer aided engineering (CAE) developers of VHDL tools, can help extend the system design capability of VHDL to simulate software behavior as an integral part of the VHDL design. Systems design normally begins with hardware, locking in design constraints before software is considered. What results is inefficient system performance. A concurrent engineering approach would maximize efficiency, reduce the design time of allocating hardware and software resources, and significantly reduce integration problems and total system cost.

Ada 95 is the new draft revision of the Ada programming language (ANSI/MIL-STD 1815) scheduled to be an ISO and ANSI standard in January 1995 (ANSI/ISO/IEC-8652:1995). Ada 95 provides a tremendous amount of new capabilities such as object oriented programming, hierarchical libraries to reduce recompilation, and several real time features to increase program efficiency. To reuse existing programs in other programming languages, Ada 95 provides bindings in the form of standardized packages for interfacing with C, COBOL, and FORTRAN. It also provides interfacing pragmas including Import, Export, and Convention which can be used with subprogram calls to other languages. This report incorporates these interfacing pragmas in its bindings from Ada 95 to VHDL. Ada 95 was selected by the author of this report because of its tremendous value to large programming efforts (both commercial and military) and because it is expected to be competitive with other software languages as compilers become available. Five years on the Ada 95 Distinguished Reviewer Team and ten years evaluating the VHDL language development and CAD tools led the author to develop a VHDL to Ada 95 binding as a first step toward concurrent engineering. Since both VHDL and Ada 95 are newly revised, taking advantage of these new language capabilities should maximize the useful lifetime of such bindings.

At the time this report was written, VHDL just finished a new revision (June 1994). Ada 95 existed as a draft ISO standard but later became a revised Ada standard in January 1995.

3. VHDL Provided Language Interface Feature

The following VHDL features are used in the VHDL to Ada 95 binding. These package and associated rules come from the VHDL standard (ANSI/IEEE Std 1076-1993):

The following example uses the VHDL attribute FOREIGN (from ANSI/IEEE Std 1076-1993, para 2.2 (230), page 24) to declare a foreign function subprogram.

```
package P is
      function F return INTEGER;
      attribute FOREIGN of F: function is
              "implementation-dependent information";
end package P;
```

Notes specific to binding [1]:

a. Parameter passing mechanisms for foreign subprograms are not defined by VHDL but can be specified in Ada 95.

b. Pure subprograms are pure functions that have no effect on other objects in the description except for computing the returned value.

c. Special VHDL elaboration rules (Sections 12.3 and 12.4) are repeated here for convenience to the reader.

The elaboration of a declarative part consists of the elaboration of the declarative items, if any, in the order in which they are given in the declarative part. This rule holds for all declarative parts, with three exceptions [1]:

1. The entity declarative part of a design entity whose corresponding architecture is decorated with the 'FOREIGN attribute defined in package STANDARD.

2. The architecture declarative part of a design entity whose architecture is decorated with the 'FOREIGN attribute defined in package STANDARD.

3. A subprogram declarative part whose subprogram is decorated with the 'FOREIGN attribute defined in package STANDARD.

For these cases, the declarative items are not elaborated; instead, the design entity or subprogram is subject to implementation-dependent elaboration.

(Note: The value of every signal will be defined by the time simulation begins.)

Summary of other elaboration rules:
You can't use a declaration until it is elaborated.

d. A pure function subprogram may not reference a shared variable.

e. Package STANDARD (Section 14.2) includes:

attribute FOREIGN: STRING;

Page 202 of package STANDARD states:

The VHDL 'FOREIGN attribute may be associated only with architectures (see Section 1.2) or with subprograms. In the latter case, the attribute specification must appear in the declarative part in which the subprogram is declared (see Section 2.1).

f. The STRING value of the attribute may specify implementation-dependent information about the foreign subprogram. The subprogram describes an implementation defined action. Foreign subprograms may have non-VHDL implementations.

g. An implementation may place restrictions on the allowable modes, classes, and types of the formal parameters to a foreign subprogram; such restrictions may include restrictions on the number and allowable order of the parameters.

h. At return, copy back of formal to actual parameters occurs (when necessary).

Note: The above VHDL implementation defined elaboration rules of 'FOREIGN attribute allow Ada 95 library units that have been pre-elaborated (**pragma** Pre-elaborate **[(***library_unit*_name**)]**;)
to be referenced through the VHDL 'FOREIGN attribute.

4. VHDL to Ada 95 Binding - BNF description

The following VHDL to Ada 95 subprogram calls use the VHDL 'FOREIGN attribute as shown above and adds Ada 95 specific syntax constraints within the quotes provided by the 'FOREIGN declaration. Signals and Files are not represented as actual parameters of the subroutine and function calls because they are not defined in the Ada 95 language. The designer would only call Ada 95 programs for behavior that the Ada 95 language can provide. Therefore, signals and files would be confined to VHDL. Buffer and linkage modes are also not defined in the Ada 95 language. Therefore, the only modes specified are **in**, **out**, and **inout**. (Note: Ada 95 expresses mode **inout** as **in out**.) The string within the quotes represents Ada 95 specific information. Since Ada 95 allows for child and parent procedures within Ada 95 hierarchical libraries, procedure and function names may refer to child or parent procedures or functions. Also, Ada 95 parameter specs may include access types with the reserved word **access** before the subtype name.

```
subprogram_specification ::=
   procedure  identifier | operational symbol
      [({ [constant] identifier, {identifier} :in
                        subtype_indication
        | [variable] identifier, {identifier} : [ in  | out |
   inout ] subtype_indication [:= static_expression ] })];

      attribute FOREIGN of identifier : procedure is
     "procedure [parent_unit_name.] identifier [formal_part]";
```

The BNF description of a function call with binding information is:

```
[ pure | impure ] function   identifier | operator
      [({ [constant] identifier, {identifier} :in
                        subtype_indication
        | [variable] identifier, {identifier} : [ in  | out |
   inout ] subtype_indication [:= static_expression ] })];

      attribute FOREIGN of identifier : function is
     "function [parent_unit_name.] identifier [formal_part]";
```

The formal_part is described as follows:

```
formal_part ::= ( parameter_spec { parameter_spec }

parameter_spec ::=
     identifier {, identifier} : [ in ] | in out | out
          subtype_name [ := default_expression ]
   | identifier {, identifier} : access subtype_name
          [ := default_expression ]
```

Elaboration is accomplished by Ada 95 rules, which can be controlled by an Ada 95 pre-elaboration pragma if needed.

5. *Package VHDL_to_Ada_95*

Since one of the original development goals of VHDL was to be Ada like when appropriate, many of the VHDL types are the same as Ada. However, some type differences exist. The following type declarations provide those Ada 95 types unsupported by VHDL which could return as parameters through the VHDL to Ada 95 interface. No subprograms using these types are provided in this package. These types should only be used by subprograms within the Ada 95 language. These types are declared in this package for storage purposes or for assigning values within the VHDL language.

Ada 95 contains type real which consists of floating point and fixed point types. VHDL contains type real which only supports floating point but not fixed point (at least not without the following declaration). Ada 95 defines floating point using the reserved word **digits**. VHDL has no reserved word **digits** but instead, is implementation defined with a minimum range. Ada 95 fixed point is defined using the reserved word **delta**. Again, VHDL has no such reserved word. Therefore, fixed point should be avoided when using this VHDL_to_Ada_95 interface.

If the Ada 95 optional Numerics Annex is implemented, then different models for floating and fixed point types are provided by Ada 95. However, there still is a problem in finding a VHDL equivalent for fixed point.

```
package VHDL_to_Ada_95 is


     -- type Fixed_point :  -- No way of expressing Fixed Point
     --                        in VDHL has been found.

     -- type derived  (derived types are a proposed addition to
     --               a future revision of VHDL). Without this
     --               or some alternative OOP approach, inheritance
     --               across the language interface is not
     --               supported with this package [4].


     -- The following are example calls through interface
     -- procedures or using interface functions.

     -- procedure call:
--   procedure Name (variable Name : inout Type := Initial_value
--   attribute FOREIGN of Name : procedure is
-- "procedure Ada_name (Name : in out Type := Initial_value)"

     -- function call:
--   function Name return String;
--   attribute FOREIGN of Name : function is
-- "function Ada_name return String;"
```

```
end package VHDL_to_Ada_95;
```

When implementing a package like the above, type integer accuracy should be taken into consideration. VHDL integer accuracy is guaranteed to have a range of -2147483647 to +2147483647. Ada 95 integer accuracy shall have at least the range -2**15 + 1 to 2**15 -1. If the Ada 95 Numerics Appendix which includes complex arithmetic with functions and procedures is used, then accuracy requirements are different.

Many attributes are different between VHDL and Ada 95. Therefore, using attributes of returned parameters from the interface to Ada 95 should be avoided.

The type conversion procedures and status functions in the above package are provided to convert Ada 95 types which are incompatible to corresponding VHDL types. Incompatable types could be returned through parameters within the VHDL to Ada 95 interface.

6. *VHDL to Ada 95 Binding - Example*

The following is an example of a procedure call and a function call using the new VHDL FOREIGN attribute.

```
package P is
     -- procedure call:
 procedure New_Car (variable Passenger : inout Person
 attribute FOREIGN of New_Car: procedure is
"procedure Vehicle.New_Car (Passenger : in out Person)"

     -- function call:
 function Attitude return String;
 attribute FOREIGN of Attitude : function is
"function Attitude return String;"
end package P;
```

The following example shows a procedure call from a VHDL process to an Ada 95 procedure. The call is indirrect through a VHDL package. The Ada procedure implements the behavior of a calculation. In contrast to this example is an equivalent behavior which is implemented into another VHDL process. Both processes are embedded in architectures.

This example, although quite simplified, illustrates how architectures containing VHDL processes and those containing foreign procedure calls to Ada 95 procedures might be selected in the system being simulated. By building hardware and software behavior representations of architectures within the VHDL library, allocation of behavior within the system can be simulated in either hardware (VHDL processes) or software (Ada 95 procedures, functions, tasks, protected types, or subprograms distributed across partitions) or a combination of the two.

```
package VHDL_to_Ada_95 is
     procedure A (X:in; Y:out)
     attribute FOREIGN of A procedure is
     "procedure  A_Ada  (X:in:  Y:  out);    -- declaration  of
     ...                              -- Ada 95 procedure
end package VHDL_to_Ada_95;

package body VHDL_to_Ada_95 is
     procedure A (X:in; Y:out)
     attribute FOREIGN of A procedure is
     "procedure  A_Ada  (X:in:  Y:  out);    -- declaration  of
begin                                        -- Ada 95 procedure.
     ...
end VHDL_to_Ada_95;


library ...
use VHDL_to_Ada_95.all
architecture Hardware of Entity_Com is
```

```
      begin
            process Do_it (Clk)
            begin
                  A (X,Y);
            end;
      ...
...

-- On Ada 95 side of interface:

package Ada_example is
...
procedure A_Ada (X,Y)
...
end package Ada_example;


package body Ada_example is

procedure A_Ada (X,Y)
      begin
            -- Example behavior in
            -- Ada procedure.
            Y = X + 2*X**2 + 3*X**3;
      end A_Ada;
begin
      A_Ada (X,Y);
      ...
end procedure A_Ada;
      ...
end package Ada_example;
```

Equivalent VHDL process of above Ada behavior:

```
architecture Hardware of System;
      ...
process B_VHDL
begin

      Y = X + 2*X**2 + 3*X**3;
end process B_VHDL;
```

Because the VHDL process can only provide limited functionality compaired to what Ada or Ada 95 can provide, a more realistic and useful example would contain several processes plus additional hardware providing a total function similar to what Ada 95 code can represent.

7. *Ada 95 Provided Language Features and BNF description*

In order to show what specific Ada 95 features are to be used in the Ada 95 to VHDL binding, the following pragmas come from the Ada 95 (ANSI/ISO/IEC-8652:1995) standard.

Ada 95 interfacing pragmas:

pragma Import ([Convention =>] *convention*_identifier,
[Entity =>] local_name [,[External_name =>]
*string*_expression] [,[Link_Name =>]
*string*_expression]);

pragma Export ([Convention =>] *convention*_identifier,
[Entity =>] local_name [,[External_name =>]
*string*_expression] [,[Link_Name =>]
*string*_expression]);

pragma Convention ([Convention =>] *convention*_identifier,
[Entity =>] *local*_name);

Interfacing pragmas are only allowed at a declarative_item or at a compilation unit.

link name -> expected type is string

convention name -> implementation defined
except for Ada and Intrinsic. (Ada 95 conformance rules 6.3.1)

calling convention -> convention of a callable entity.

pragma Linker_options (*String*_expression);

A pragma Linker_Options is allowed only at the place of a declarative item.

8. Package Ada_95_to_VHDL

```
package Ada_95_to_VHDL is

     type BIT is ('0','1');
     type BIT_VECTOR is array (Natural range <>) of BIT;

     -- The following example represents how each Ada 95 interface
     -- pragma can be used for interacting with other languages.

     -- pragma Import (
     --    Entity => Clock
     --     Tick);

     -- pragma Export (VHDL, ;

     -- pragma VHDL (Convention => );

end package Ada_95_to_VHDL;
```

Since one of the original goals of VHDL was to make it Ada like when appropriate, many of the VHDL types are the same as Ada. However, some type differences exist. The above type declarations provide those VHDL types unsupported by Ada 95 which could return as parameters through the Ada 95 to VHDL interface. No subprograms using these types are provided in this package. These types should only be used by subprograms within the VHDL language. These types are declared in this package for storage purposes only within the Ada 95 language. The above two types are repeated from the VHDL predefined environment standard package. They are not in the Ada predefined environment.

VHDL types unsupported by Ada 95:

VHDL **signal**s cannot be represented (at least without complication) by Ada 95 types and should not be passed across the Ada 95 to VHDL interface. However, signals can be set to values represented by constants or variables sent across the interface by parameters. The subtype of the signal has to be restricted to types compatible to Ada 95 types.

Since VHDL type **physical** requires a language syntax that is unsupported by Ada 95, no way could be found to hold physical type information, or units, in Ada 95. Therefore, type physical should be avoided when using the above package.

9. Ada 95 to VHDL Binding - Examples

The following simple example shows how the above Ada 95 pragmas could be used to call VHDL subprograms or access VHDL data types. Due to differences in type restrictions between the two languages, Ada 95 access to some VHDL types are not allowed. For example, a signal has no meaning in Ada 95. VHDL variables have limited use compared to Ada 95.

```
-- Ada 95 side of the Interface:

procedure Work_with (Bit_Stream : in out BIT_Vector) is
...
end Work_with;


pragma Import (Status_Register, Register)

Work_with (Status_Register);

pragma Export (Status_Register, Register);


-- VHDL side of interface:

package Hardware is
   type W is array (Integer range <>) of BIT;

   variable Register : W (0 to 31);
   ...
end Hardware;

with Hardware;
architecture ...
   ...
```

10. System Design Considerations

Designing a system includes allocating system functional components to hardware and software. This involves making design tradeoffs for efficiency. If the computer aided engineering system provides the capability to substitute an Ada task for a concurrent VHDL process, then the designer can simulate both from within the VHDL design to test which combination of hardware and software implementations works most efficiently.

An architecture can include processes that implement a certain functionality or, by implementing some of the concepts suggested in this report, implement calls to Ada procedures using the VHDL 'FOREIGN attribute. This is suggested in the BNF below.

```
architecture ::=
     architecture Example of Entity_E is
          ...
          process ...      vs.     call_to_Ada_Subprogram
                                   that has been declared
                                   with attribute 'FOREIGN.
```

Section 11 contains examples of calling concurrent Ada 95 features from VHDL processes. Since VHDL architecture behavior is represented by concurrent process statements and other concurrent statements, tradeoffs could be made by simulating various combinations of concurrent features both in VHDL and Ada 95.

Section 12 lists Ada 95 features which tie (or map) how software interacts with hardware or resides with hardware in the system. These features interface coexisting hardware (or VHDL representations of hardware) and software (implemented in Ada 95).

11. VHDL processes interacting with Ada 95 concurrent features

The following discussion describes how VHDL processes can use the VHDL to Ada 95 interface to activate Ada 95 tasks and other concurrent features.

a. Interacting with Ada tasks

A VHDL process passes a constant or variable through a VHDL subroutine parameter. The subroutine in this interface package calls an Ada 95 subprogram which activates a task and passes appropriate parameter values to the task's accept statement (or task entry). The Ada 95 task controls the VHDL process passing a variable to the Ada 95 interface subprogram which passes it to the VHDL interface subprogram. A signal is assigned the current value of the variable within a VHDL subroutine. The signal then controls the process through the process sensitivity list. Therefore, the desired behavior is for the process to wait for the Ada task to complete before it continues.

The following are example interactions between a VHDL process and an Ada 95 task.

Implementing interface package to activate an Ada 95 task:

```
package VHDL_to_Ada_95 is
     procedure Synchronize (X:in; Y:out)
     attribute FOREIGN of Synchronize: procedure is
     "procedure Activate (X:in: Y: out);  -- declaration of Ada 95
                              -- procedure that activates Ada 95 task.
          ...
          ...
end package VHDL_to_Ada_95;


package body VHDL_to_Ada_95 is
     procedure Synchronize (X,Y);
     ...
     ...
end VHDL_to_Ada_95;


library ...
use VHDL_to_Ada_95.all
architecture Hardware of Entity_Com is
     begin
          process Do_it (Clk)
          begin
               Synchronize (X,Clk);
               ...
          end;
     ...
```

```
...

-- Ada 95 side of interface:

procedure Activate (X,Y) is
   begin
      ...
      Sync (X,Y);
      ...
      end Activate;


task Task_A is
      entry Sync (X,Y);
      begin
      ...
      end;
...
task body Task_A is
...
begin
      ...
      loop
         select
            when ...
            accept Sync (X: in) do
            ...
            end Sync;
         or
            when ...
            accept ...
            ...
            end ...
         end select;
end Task_A;
```

b. Interacting with Ada 95 protected types

The following interactions provide VHDL process to Ada 95 protected types.

```
--    with the above VHDL code:
      procedure Activate (X,Y)
         begin
            Set_Component (X,Y);
                  ...
         end procedure Activate;
--    continue with Ada 95 code.  The following protected
--    type declaration and body is taken from the Ada 95
--    reference manual [2] for illustrative purposes on
--    what is called from the above procedure.

      protected Shared_Array is
         function component (N: in Index) return Item);
```

```
      procedure Set_Component (N: in Index; E: in Item);
   private
      Table : Item_Array (Index) := (others => Null_Item);
   end Shared_array;

   protected body Shared_Array is
      function Component(N : in Index) return Item is
      begin
         return Table(N);
      end Component;

   procedure Set_Component(N : in Index; E : in Item) is
      begin
         Table(N) := E;
      end Set_Component;
   end Shared_Array;

--    end of example from Ada 95 reference manual.
```

c. Interacting with Ada 95 asynchronous control features

The following interactions provide VHDL process to real-time Ada 95 synchronous task control for semaphore (or suspended objects) two stage suspend operations. Ada 95 compiler implementation of the Real-Time Systems Annex is required for this capability.

The following procedure calls to the Asynchronous Task Control package in the Ada 95 Real Time Systems Annex:

```
   procedure Activate (X)
      begin
         Hold(X);  -- declared in the Ada95 environment
         ...
      if Is_Held, then  -- A function declared in Ada 95
         Continue (X);  -- A procedure declared in Ada 95
      end Activate;
```

The following procedure activates the Asynchronous Select (in a task with an abortable part):

```
-- Ada procedure
-- called from VHDL
procedure Activate
   begin
      Async(X);
      ...
   end Activate;
```

```
-- Ada 95 Asynch Select:
(limited asynch behavior)
select
   accept ASync(X);
   ...
then abort
   ...
end select;
```

d. Interacting with Ada 95 distributed processing features

The following interactions provide VHDL processes access to remote procedure calls (RPCs) across Ada 95 partitions for distributed processing application. Ada 95 compiler implementation

of the Distributed Systems Annex is required for this capability.

Using the Remote Procedure Call (RPC) package in the System library of Ada 95 distributed annex [2]:

```
with Ada.Streams
with System.RPC

-- From within Activate procedure:
System.RPC.Read(Stream, Item,Last);
...
System.RPC.Write(Stream,Item);
...   -- Synchronous call to remote procedure
System.RPC.Do_RPC(Partition, Params, Result);
...   -- Asynchronous call to remote procedure
System.RPC.Do_APC(Partition, Params);
```

12. Mapping Ada 95 Software directly to hardware

The following is taken from the Ada 95 reference manual to show what is available for mapping the software directly to hardware or simulated hardware in VHDL.

Ada 95 representation items including representation clauses, component clauses, and representation pragmas specify how types and other features of the Ada 95 language are to be mapped onto the underlying machine. When used for interfacing to VHDL, these representation items can map Ada 95 behavior features on to VHDL simulated hardware.

The Record layout aspect of representation consists of the storage places for some of all components of a record. For example,

```
for Program_Status_word use
   record
      System_Mask      at 0*Word range 0..7;
      Protection_Key   at 0*Word range 10..11;
      ...
   end record;
```

Other examples of representation items [from 2]:

```
type Bit_Order is (High_Order_First, Low_Order_first);
Default_Bit_order : constant Bit_Order;

type T_Address is implementation_specific

pragma Convention(Intrinsic, ...);

-- Using package storage pools for storage management:
System.Storage_Pools.Allocate(Pool,...,...);

-- Use streams for generic IO through attributes:
T'READ(...);
T'WRITE(...);
-- (The above is similar to the VHDL the 'FOREIGN attribute.)
```

-- Interrupts:
Protected types can be used to control the servicing of priority interrupts.

-- Pre-elaborate:
An Ada procedure embedded in hardware such as a co-processor can be called by an Ada 95 subprogram by using the pre-elaboration pragma. This tells the compiler that the embedded procedure is already elaborated.

Appendix A Expanded BNF Descriptions of Key VHDL and Ada (95) Features

These BNF descriptions are derived from what is listed in the VHDL and (draft) Ada 95 reference manual [1 and 2].

VHDL Process statement (could be substituted for Ada task).
Guards: Sensitivity list of VHDL processes
Accept statements of Ada tasks

```
process_statement ::=
    [ process_label :]
        [ postponed ] process [( sensitivity_list )] [ is ]
            process_declarative_part
        begin
            process_statement_part
        end [ postponed ] process [ process_label ];

    sensitivity list ::=
        signal_name {, signal_name }

        Note: When sensitivity list is present, an implicit
        wait statement appears as the last statement in a
        process statement.

    process_declarative_part ::=
        { process_declarative_item }

    process_declarative_item ::=
            subprogram_declaration
        |   subprogram_body
        |   type_declaration
        |   subtype_declaration
        |   constant_declaration
        |   variable_declaration
        |   file_declaration
        |   alias_declaration
        |   attribute_declaration
        |   attribute_specification
        |   use_clause
        |   group_type_declaration
        |   group_declaration

    process_statement_part ::=
        { sequential_statement }
            Note: A process statement is "passive" if it
            contains no signal assignment statement.

    sequential_statement ::=
            wait_statement
        |   assertion_statement
        |   report_statement
        |   signal_assignment_statement
        |   variable_assignment_statement
```

```
        |  procedure_call_statement
        |  if_statement
        |  case_statement
        |  loop_statement
        |  next_statement
        |  exit_statement
        |  return_statement
        |  null_statement
```

Ada (95) Tasks contain the following:

```
handled_sequence_of_statements ::=
      sequence_of_statements
      [ exception
        exception_handler
       { exception_handler }]

sequence_of_statements ::=
         abortable_part
      |  accept_alternative
      |  case_statement_alternative
      |  conditional_entry_call
      |  delay_alternative
      |  entry_call_alternative
      |  exception_handler
      |  handled_sequence_of_statements
      |  if_statement
      |  loop_statement
      |  selective_accept
      |  triggering_alternative

exception_handler ::=
      handled_sequence_of_statements

sequence_of_statements ::=
      statement { statement }

statement ::=
        {label} simple_statement
      | {label} compound_statement

simple_statement ::=
         null_statement
      |  assignment_statement
      |  goto_statement
      |  return_statement
      |  requeue_statement
      |  abort_statement
      |  code_statement
      |  exit_statement
      |  procedure_call_statement
      |  entry_call_statement
      |  delay_statement
      |  raise_statement

compound_statement ::=
```

```
            if_statement
          | case_statement
          | loop_statement
          | block_statement
          | accept_statement
          | select_statement

    assignment_statement ::=
         variable_name := expression;
```

VHDL Subprograms:

```
    subprogram_declaration ::=
         subprogram_specification ;

    subprogram_specification ::=
              procedure designator [(formal_parameter_list)]
         |[pure | impure] function designator
         [(formal_parameter_list)]  return type_mark

    designator ::= identifier | operator_symbol

    operator_symbol ::= string_literal

    formal_parameter_list ::= parameter_interface_list


    subprogram_body ::=
         subprogram_specification is
              subprogram_declarative_part
         begin
              subprogram_statement_part
         end [subprogram_kind] [designator];

    subprogram_declarative_part ::=
         { subprogram_declarative_item }

    subprogram_declarative_item ::=
            subprogram_declaration
          | subprogram_body
          | type_declaration
          | subtype_declaration
          | constant_declaration
          | variable_declaration
          | file_declaration
          | alias_declaration
          | attribute_declaration
          | attribute_specification
          | use_clause
          | group_template_declaration
          | group_declaration

    subprogram_statement_part ::=
         { sequential_statement }

    subprogram_kind ::= procedure | function
```

VHDL Package Declarations:

```
package_declaration ::=
      package identifier is
            package_declarative_part
      end [ package ] [ package_simple_name ];

package_declarative_part ::=
      (package_declarative_item)


package_declarative)item ::=
            subprogram_declaration
      |     type
      |     subtype
      |     constant
      |     signal
      |     shared_variable
      |     file
      |     alias
      |     component
      |     attribute_declaration
      |     attribute_specification
      |     disconnection_specification
      |     use_clause
      |     group_template
      |     group


package_body ::=
      package body package_simple_name is
            package_body_declarative_part
      end [ package body ][ package_simple_name ];

package_body_declarative_part ::=
      { package_body_declarative_item }

package_body_declarative_item ::=
            subprogram_declaration
      |     subprogram_body
      |     type_declaration
      |     subtype
      |     constant
      |     shared_variable
      |     file
      |     alias
      |     use
      |     group_template
      |     group
```

Ada (95) Subprograms:

```
subprogram_declaration ::= subprogram_specification;

subprogram_specification ::=
        procedure defining_program_unit_name
                  parameter_profile
      | function defining_designator
                  parameter_and_result_profile

subprogram_body ::=
      subprogram_specification is
            declarative_part
      begin
            handled_sequence_of_statements
      end [designator];
```

Ada Subprogram Calls:

```
procedure_call_statement ::=
        procedure_name;
      | procedure_prefix actual_parameter_part;

function_call ::=
      function_name
      | function_prefix actual_parameter_part;
```

Ada Packages:

```
package_declaration ::= package_specification;

package_specification ::=
      package defining_program_unit_name is
            { basic_declarative_item }
      [ private
            { basic_declarative_item }]
      end [[ parent_unit_name.]identifier]

package_body ::=
      package body defining_program_unit_name is
            declarative_part
      [ begin
            handled_sequence_of_statements ]
      end [[parent_unit_name.]identifier];
```

VHDL Objects and Ada (95) Objects:

VHDL Types:

```
declaration ::=
    type
  | subtype
  | object
  | interface
  | alias
  | attribute
  | component
  | group_template
  | group
  | entity
  | configuration
  | subprogram
  | package

type_declaration ::=
    full
  | incomplete

full_type_declaration ::=
    type identifier is
        type_definition ;

type_definition ::=
    scaler
  | composite
  | access
  | file

scaler_type_definition ::=
    enumeration
  | integer
  | floating
  | physical

composite_type_definition ::=
    array
  | record

file_declaration
    file identifier_list :
        subtype_indication
        [ file_open_information];

subtype_declaration ::=
    subtype identifier is
        subtype_indication;
```

Ada (95) Types:

```
basic_declaration ::=
    type_declaration
  | subtype
  | object
  | number
  | subprogram
  | abstract_subprogram
  | package
  | renaming
  | exception
  | generic
  | generic_instantiation

type_declaration ::=
    full    | incomplete
  | private_type
  | private_extension

full_type_declaration ::=
    type defining_identifier
      [known_discriminant_part]
        | task_type_declaration
      |protected_type_declaration

type_definition ::=
    enumeration
  | integer
  | real
  | array
  | record
  | access
  | derived

derived_type_definition ::=
      [abstract] new
  parent_subtype_indication
 [record_extension_part]

access_type_definition ::=
    access subtype_indication

subtype_declaration ::=
    subtype identifier is
    subtype_indication;
```

References

1. The Institute of Electrical and Electronic Engineers, Inc., IEEE Standard VHDL Language Reference Manual. ANSI/IEEE Std 1076-1993.

2. International Organization for Standardization, Ada 95 Reference Manual, The Language, The Standard Libraries. ANSI/ISO/IEC-8652:1995 International Standard, Version 6.0, January 1995.

3. Intermetrics, Inc., Ada 95 Rationale. January 1995.

4. Solid State Electronics Directorate, Wright Laboratory, Air Force Materiel Command, LtCol Michael T. Mills, Technical Report: Proposed Object Oriented Programming (OOP) Enhancements to the Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL). WL-TR-93-5025, August 1993.

5. Solid State Electronics Directorate, Wright Laboratory, Air Force Systems Command, LtCol Michael T. Mills, Technical Report: Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) Syntax and Semantics Summary. WL-TR-91-5030, June 1991.